Divide and Conquer

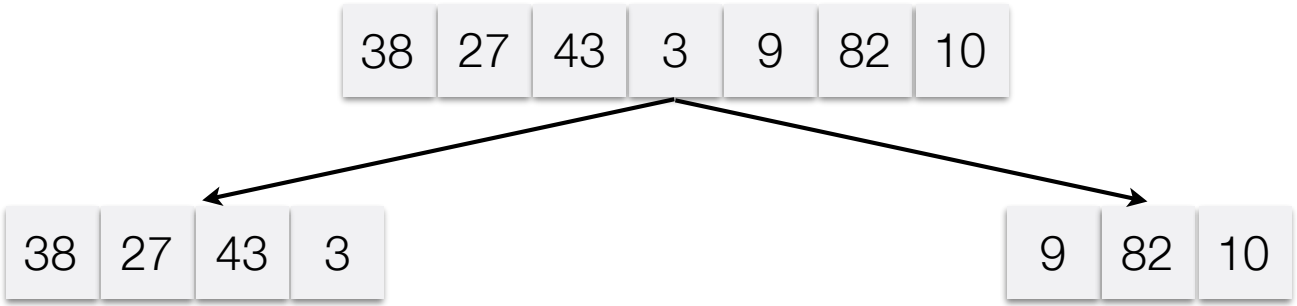
A divide and conquer algorithm recursively breaks down a problem into two or more subproblems until these have a simple direct solution. The solutions of the subproblems are then combined into a solution to the original problem.

- Efficient recursive algorithms for many problems
  - Base case = Direct solution
  - Recursive call = Split problem, solve both, recombine
- Foundation of two famous sorting algorithms
  - Merge Sort
  - Quick Sort

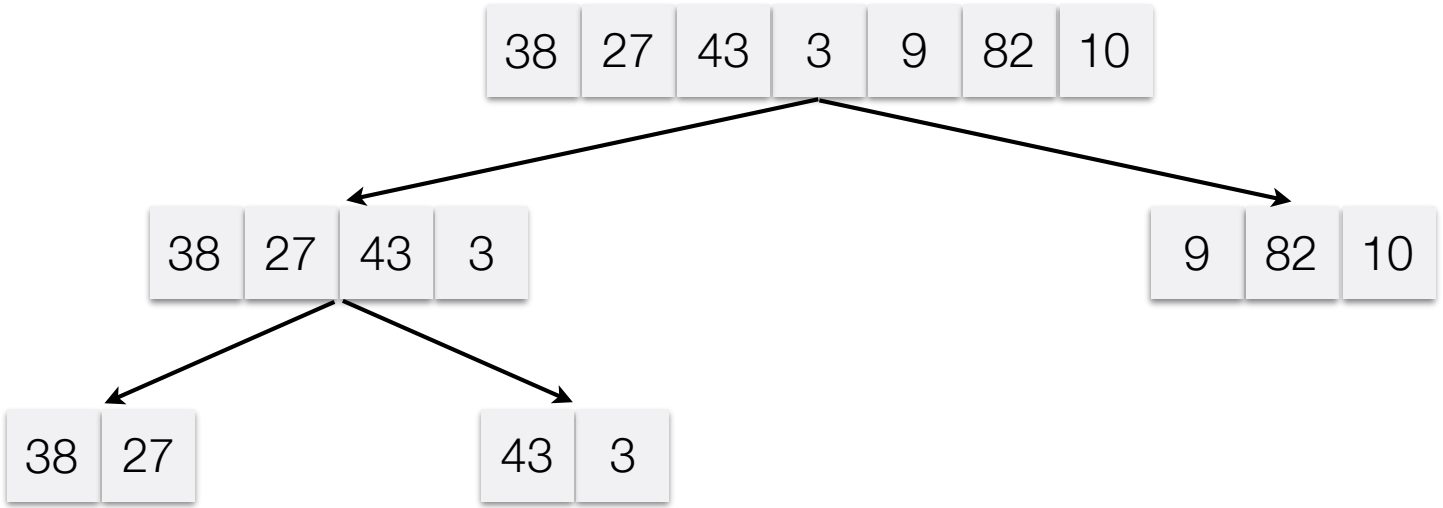
Merge Sort

- Base case
  - A single-element list is sorted
- Recursive call
  - Split the list into two halves, and merge-sort both independently
- Recombination
  - Combine the two sorted halves while maintaining (sorted) order

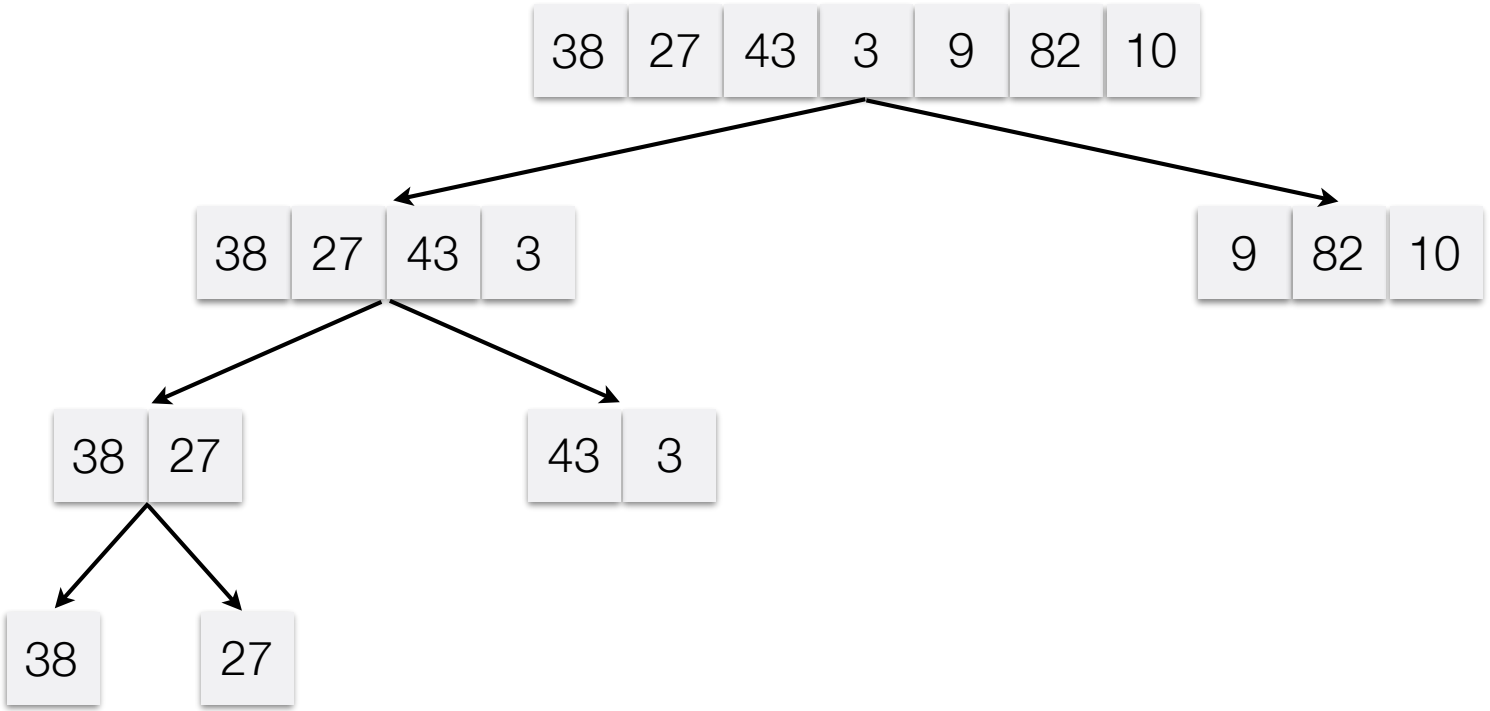

38
27
43
3
9
82
10



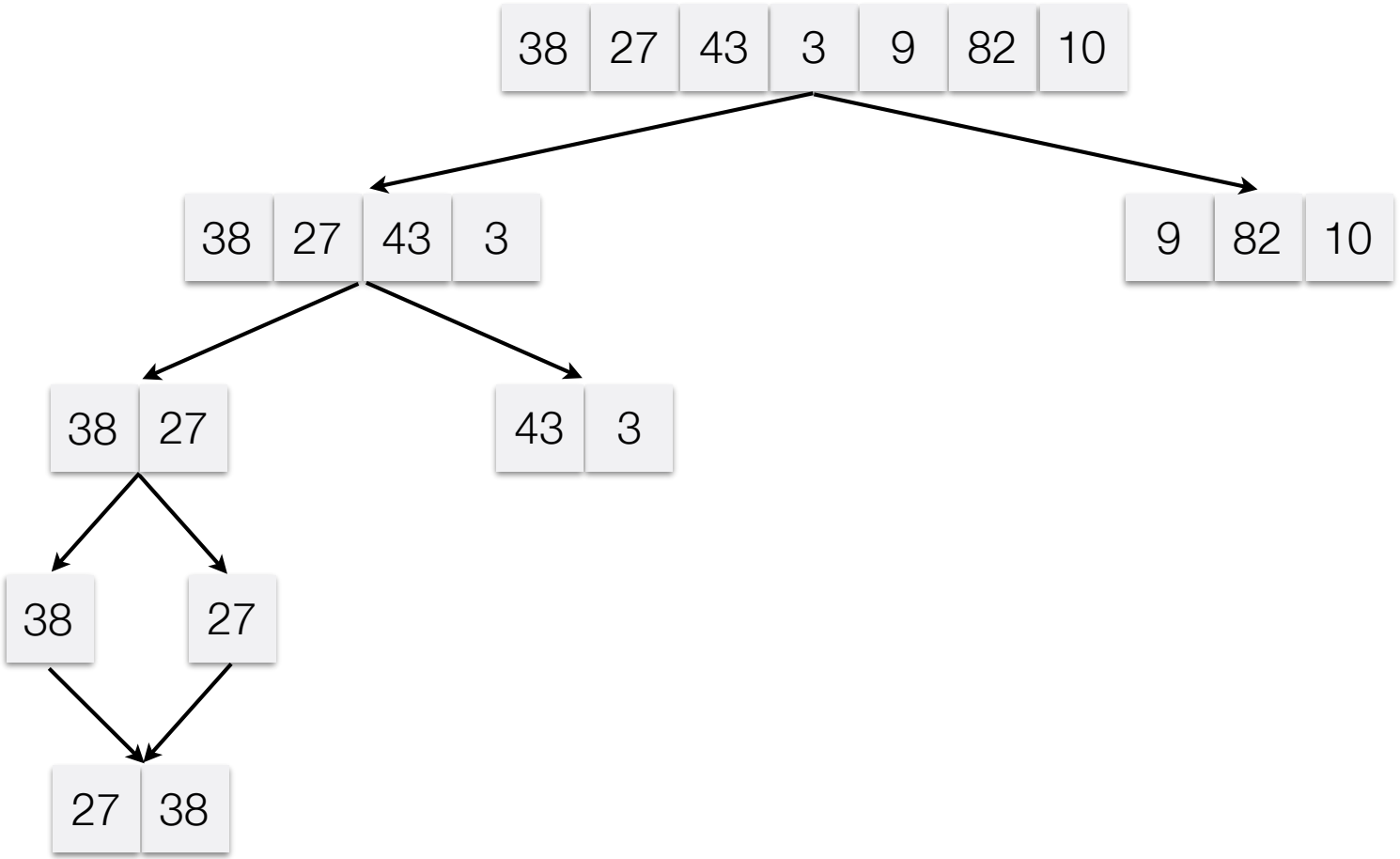
Step 1



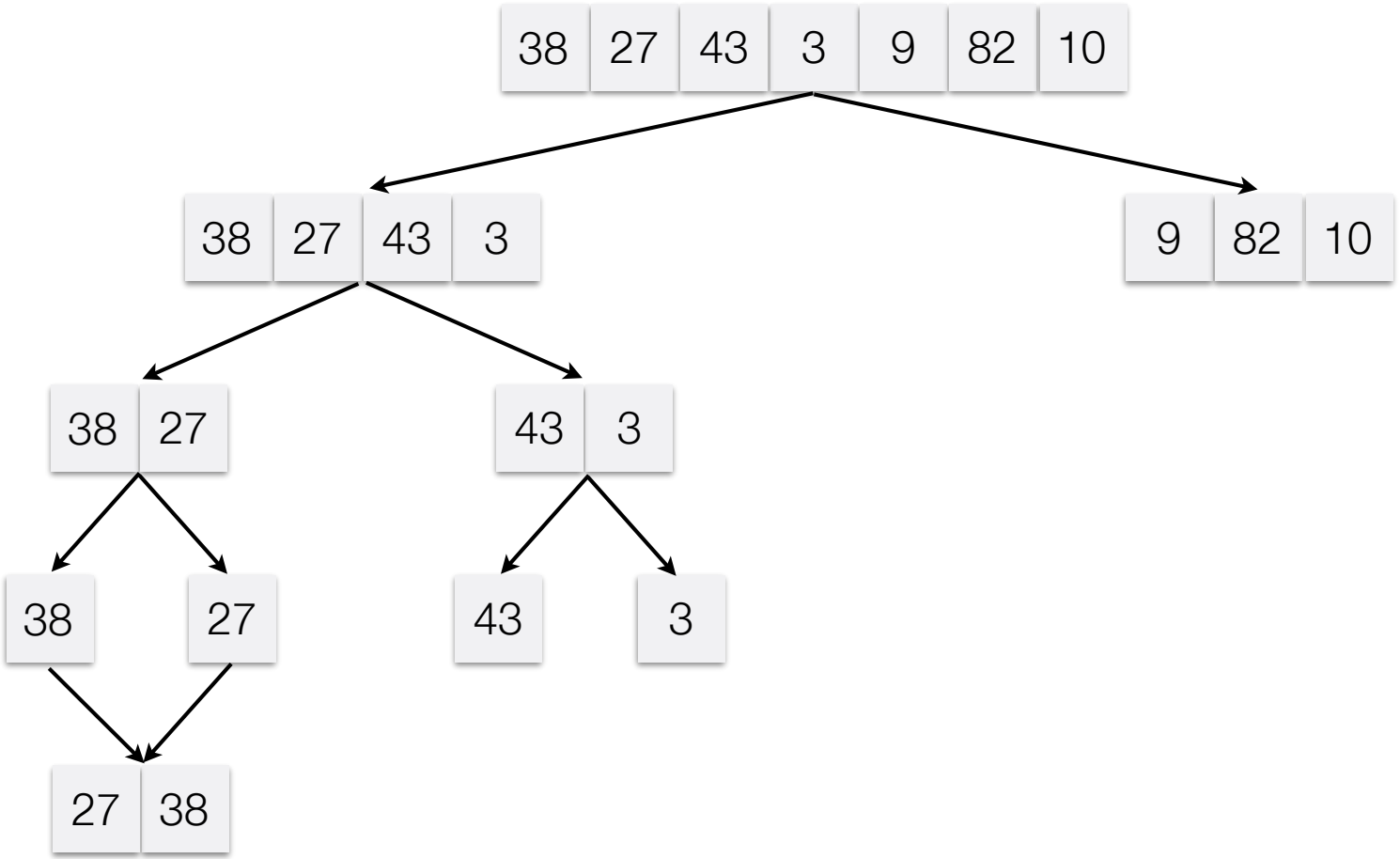
Step 2



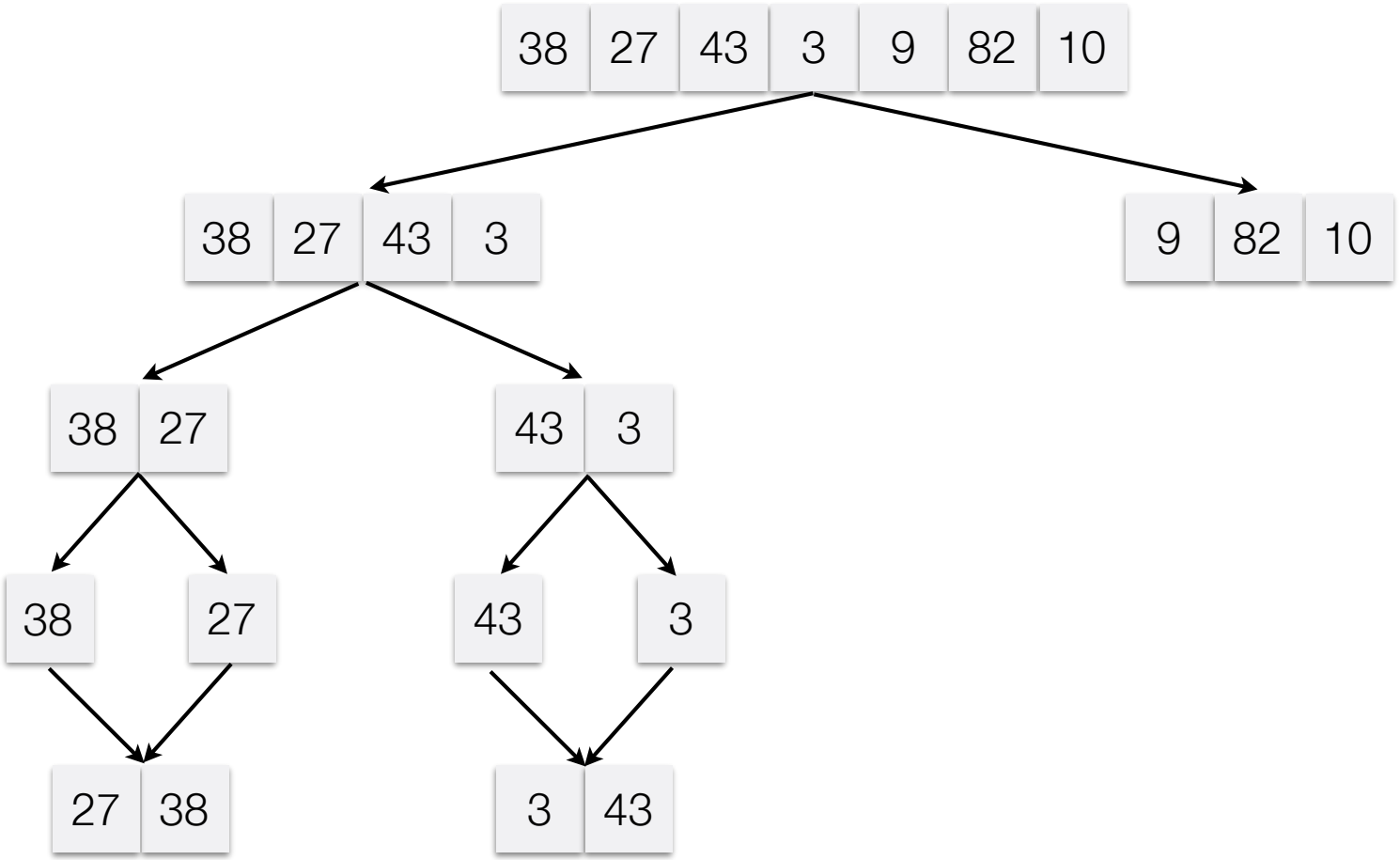
Step 3



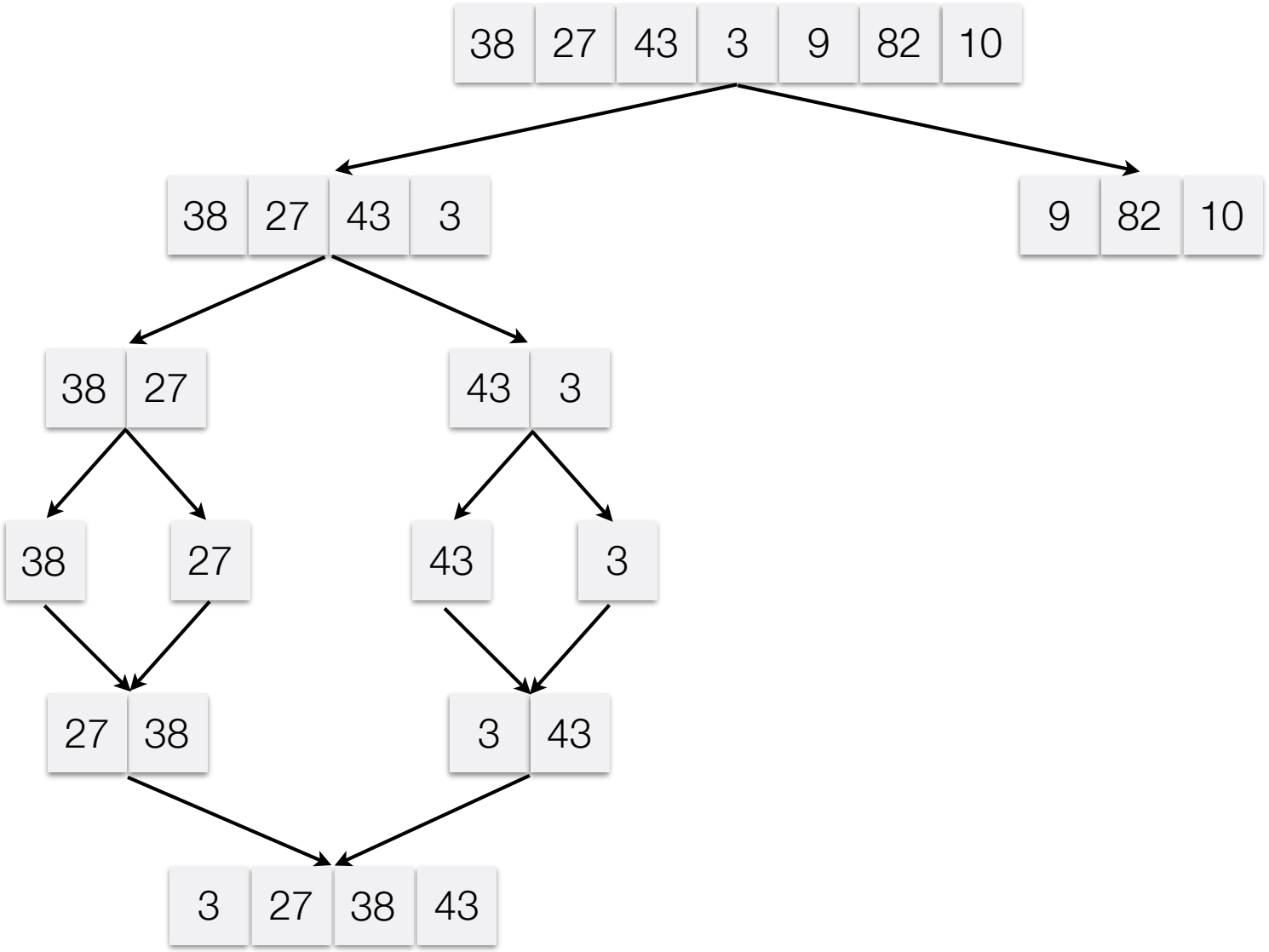
Step 4



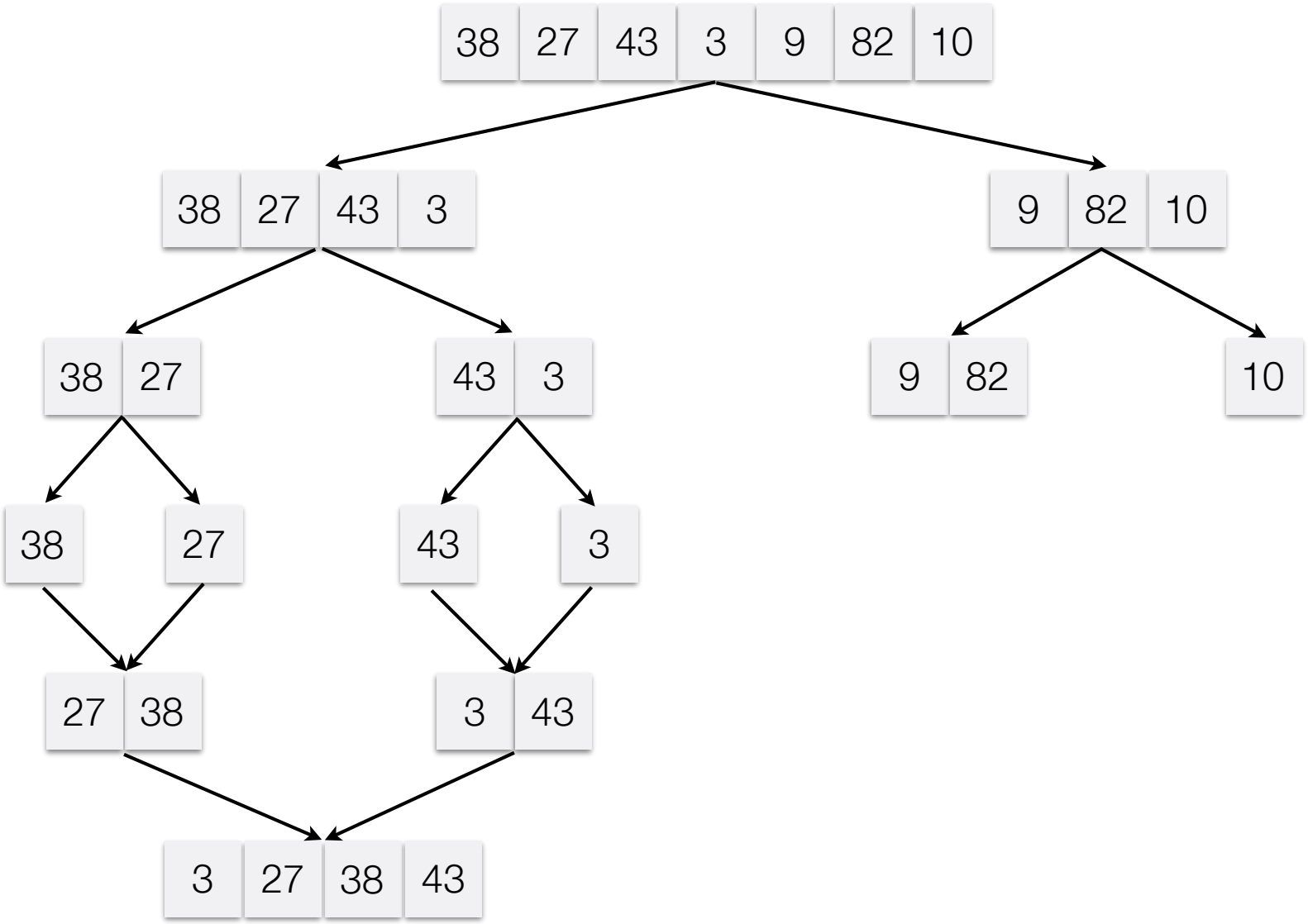
Step 5



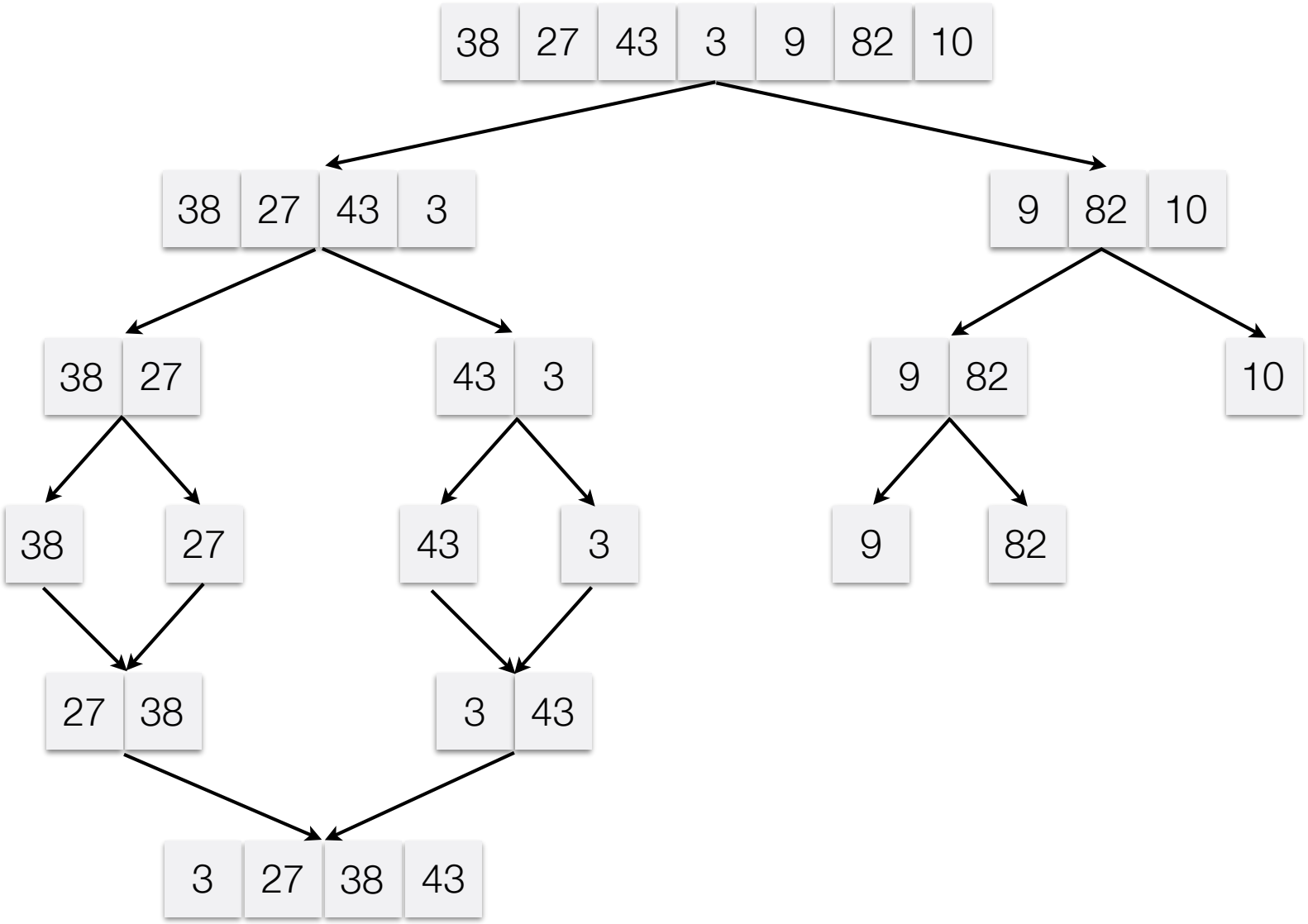
Step 6


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
38
27
43
3
27 38
3 43
3 27 38 43

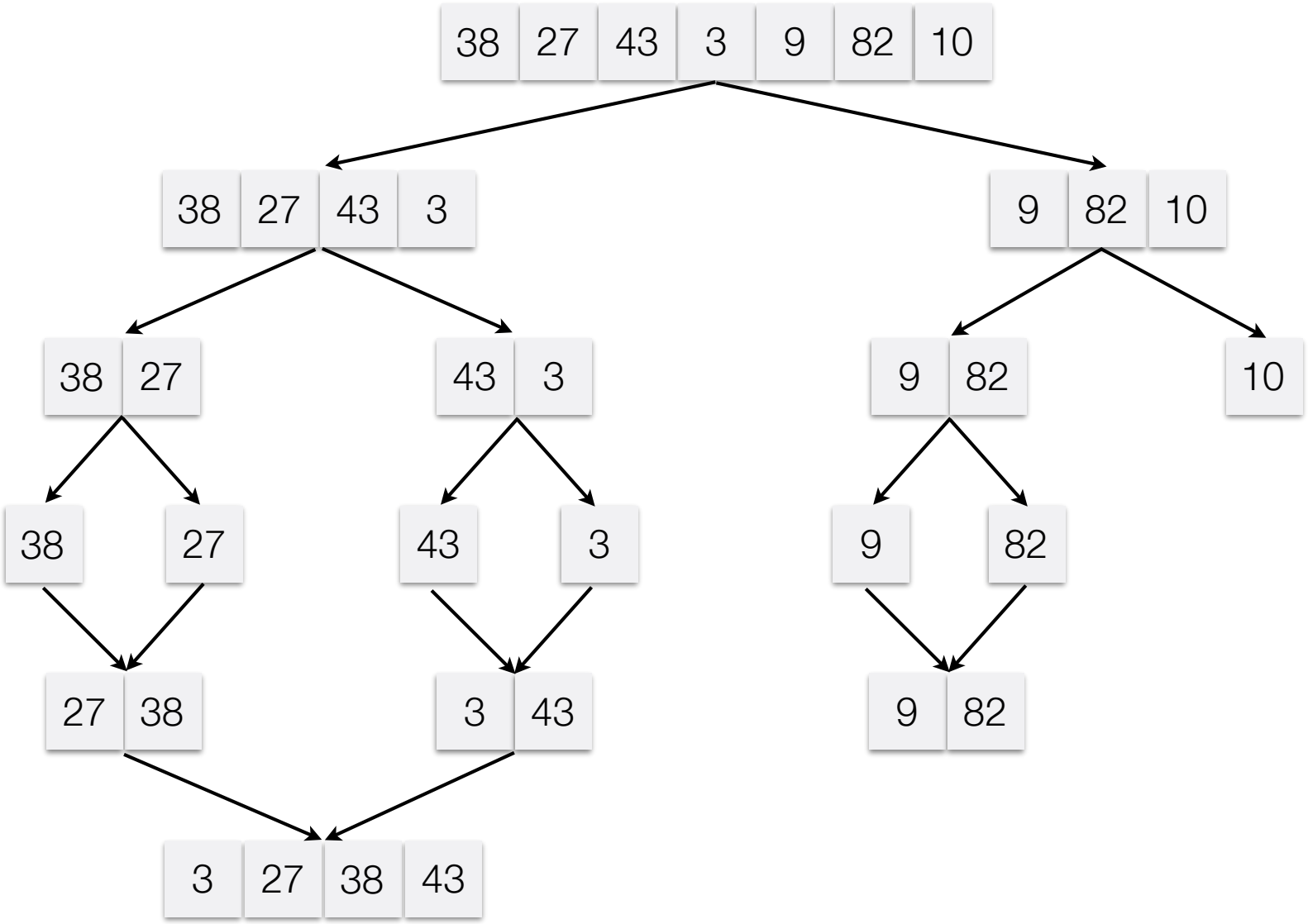
Step 7


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
27 38
3 43
3 27 38 43

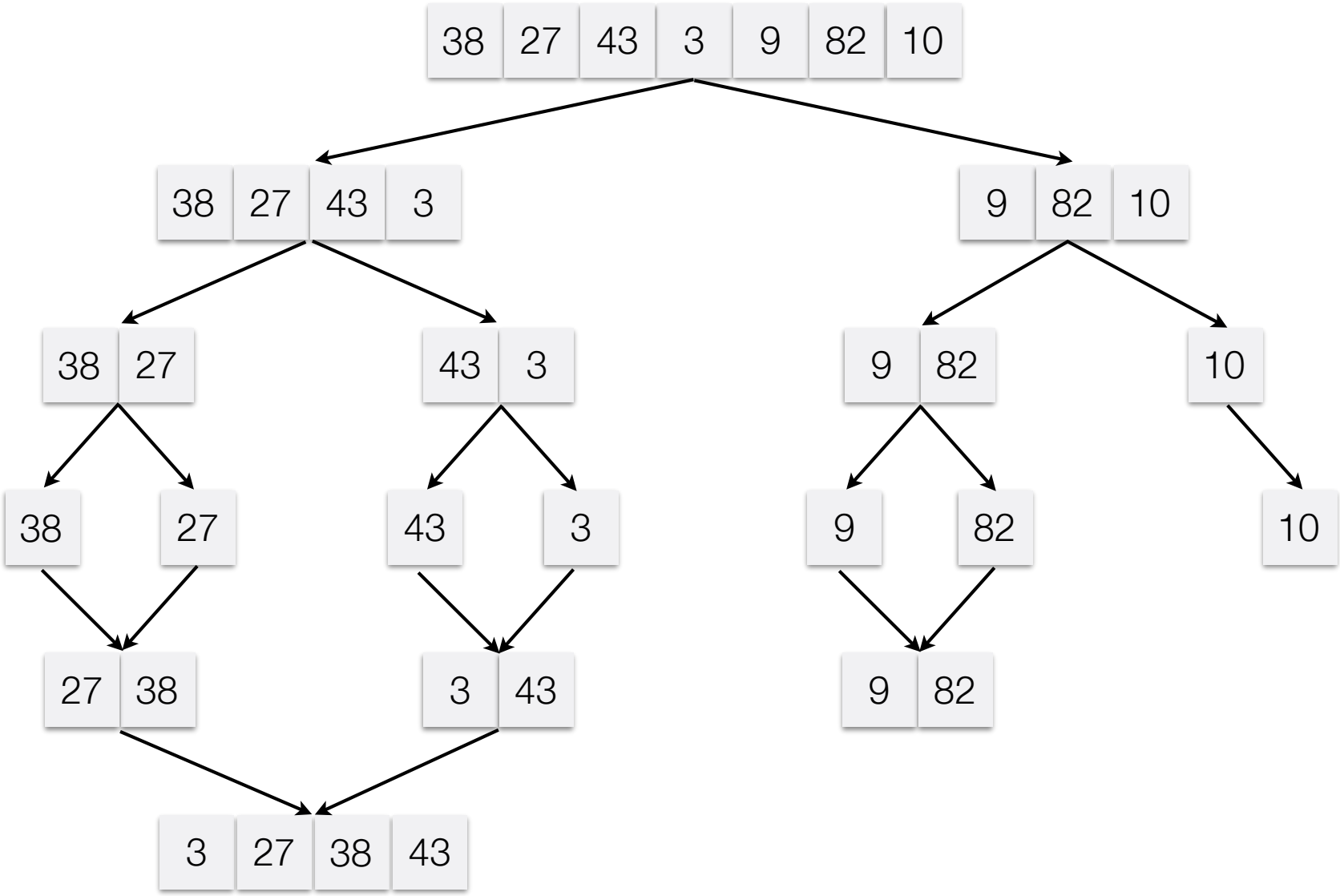
Step 8


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
27 38
3 43
3 27 38 43

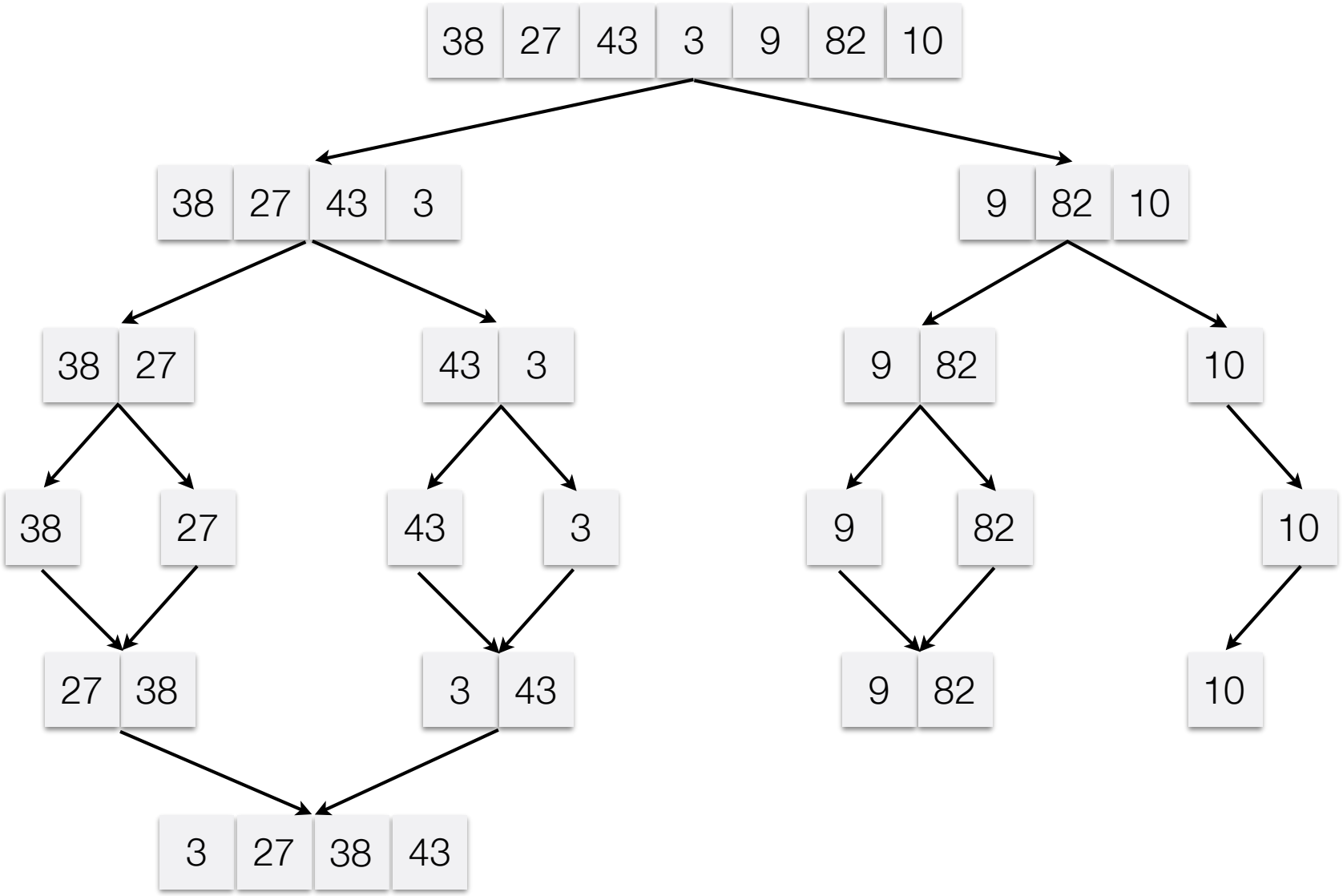
Step 9


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
27 38
3 43
9 82
3 27 38 43

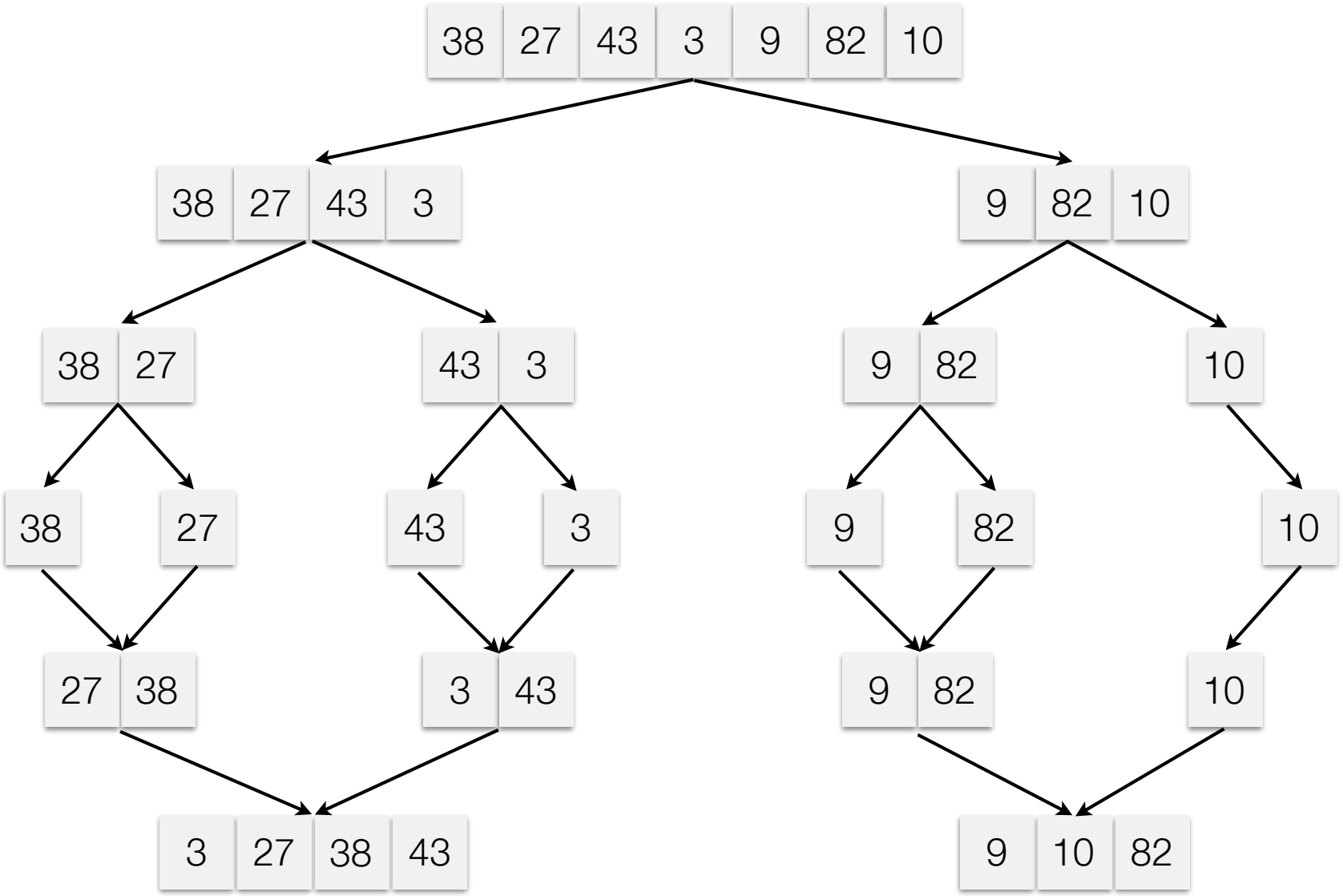
Step 10


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
3 27 38 43

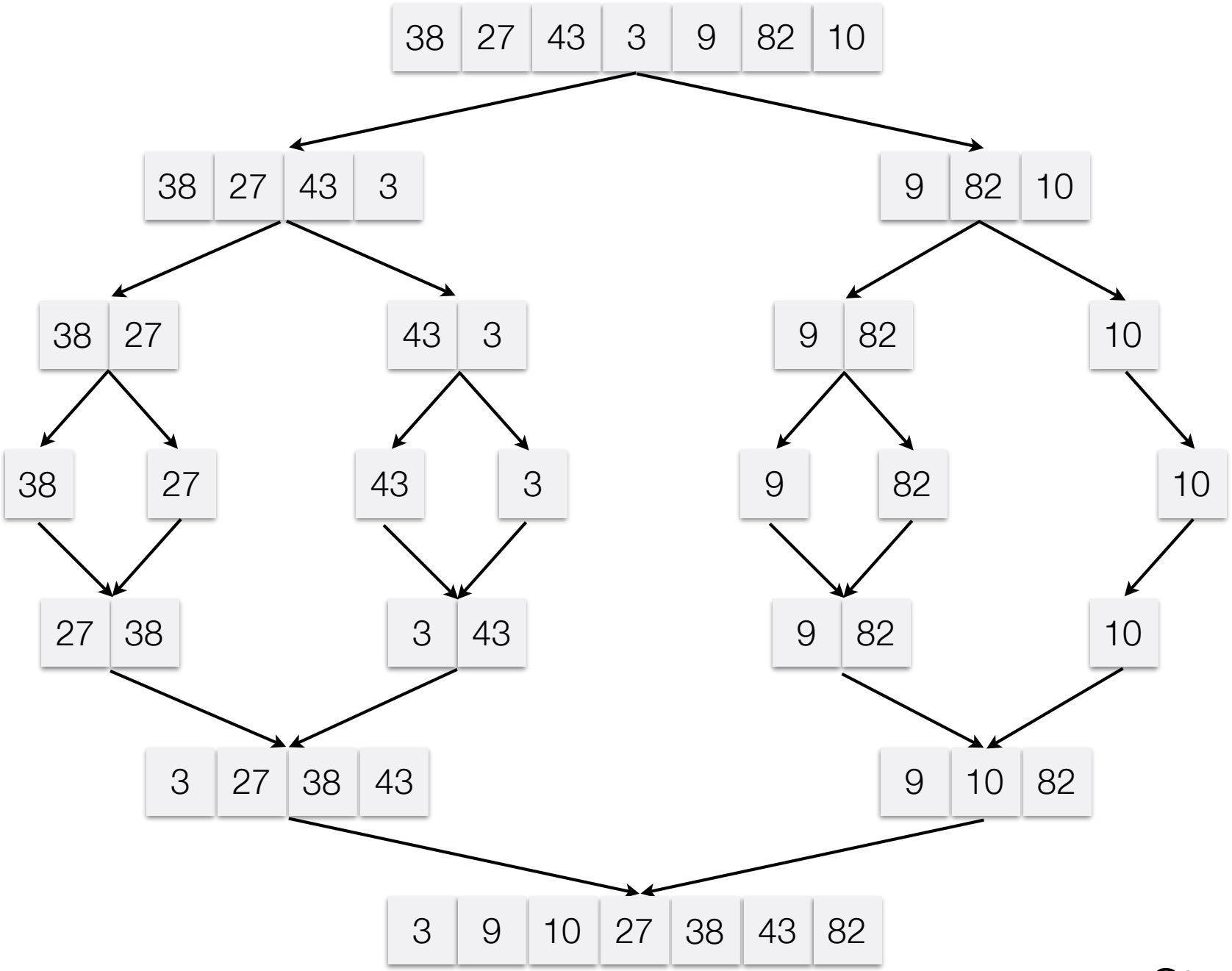
Step 11


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43

Step 12


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82

Step 13


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

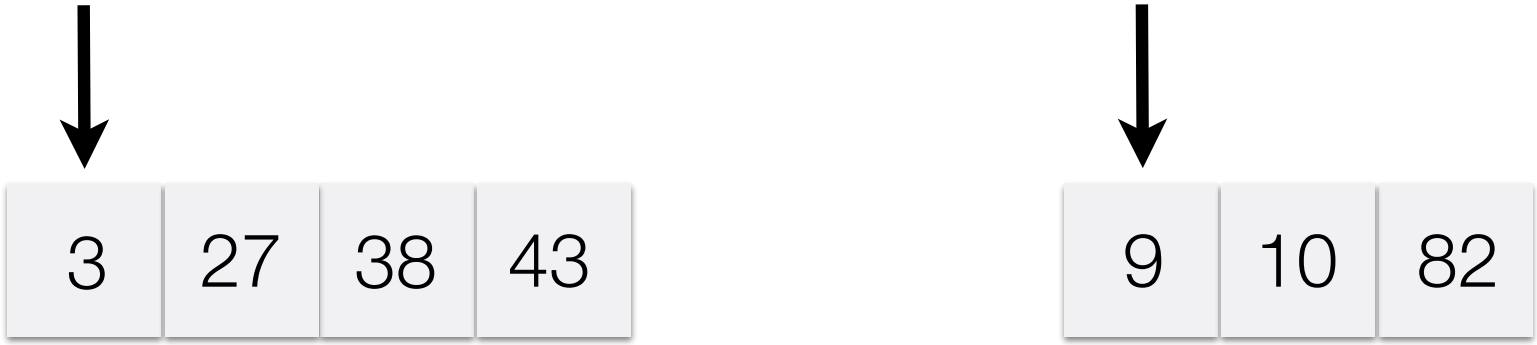
Step 13

Merging Lists

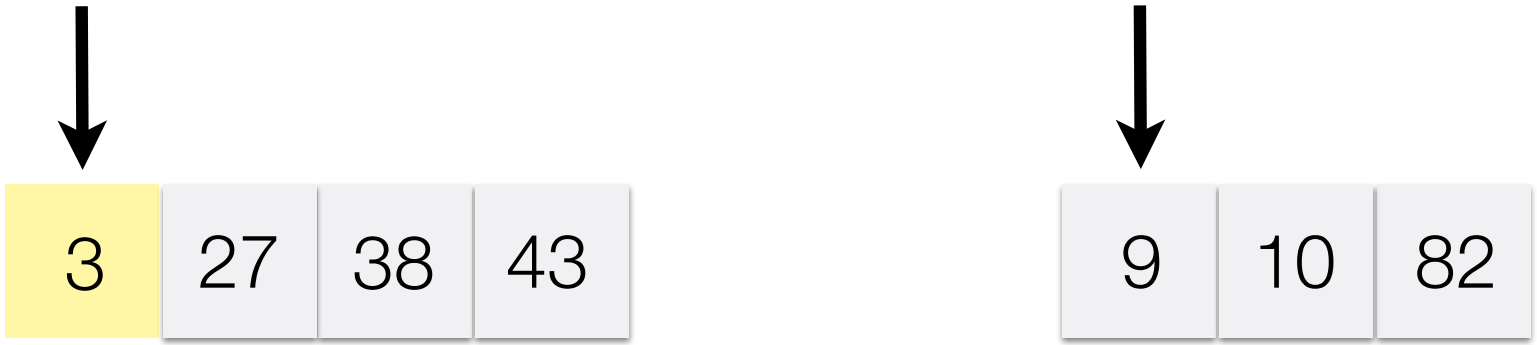
The core work of Merge sort is the merging of sublists

- Make a new list to store merged list
- Keep pointers into both source lists
    - While both pointers are not at the end of the lists
        - Compare current elements
        - Put least one in the destination list
        - Increase appropriate pointer into source lists

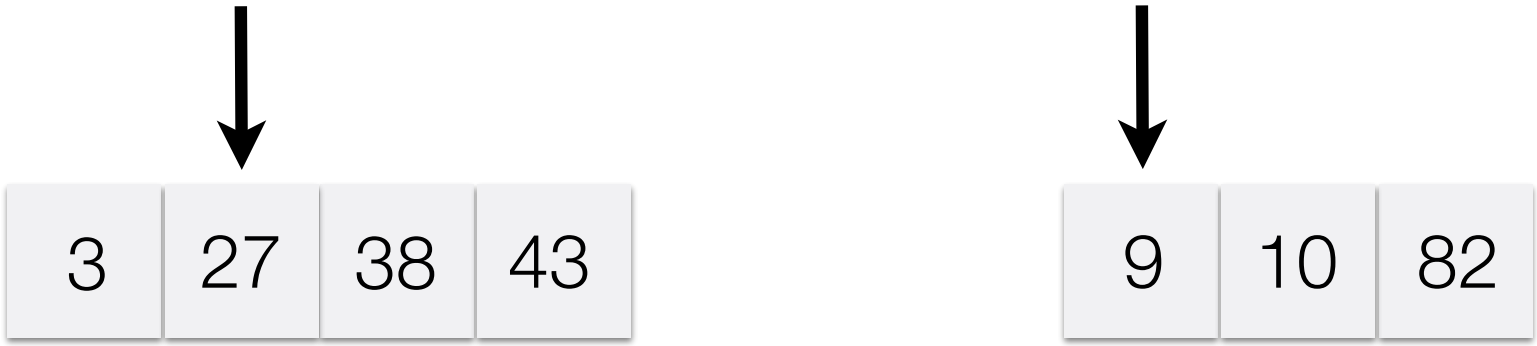
Merging Lists



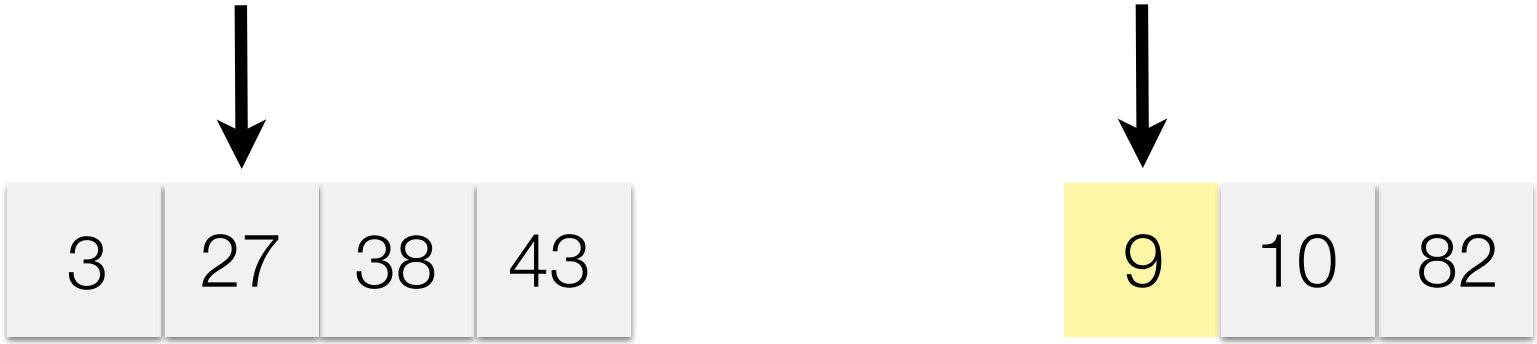
Merging Lists



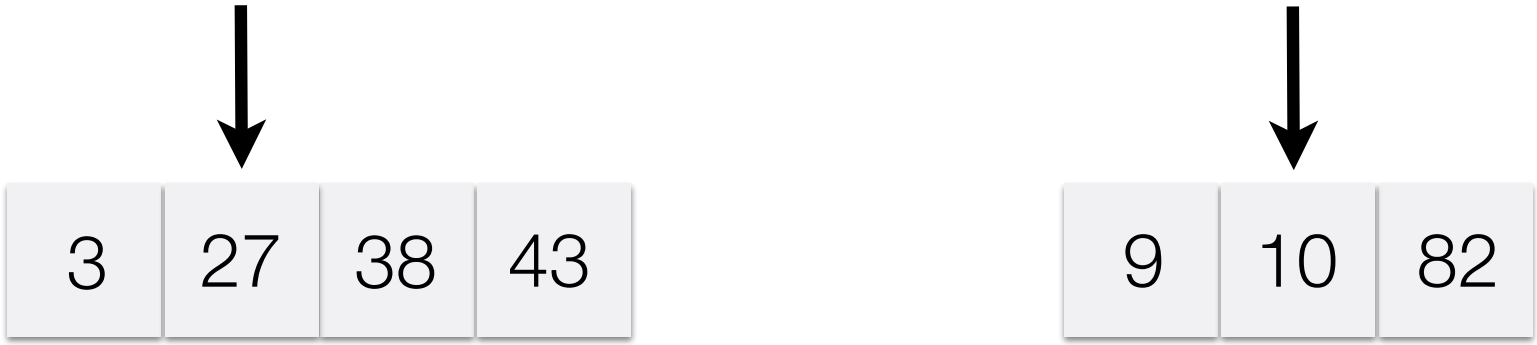
Merging Lists



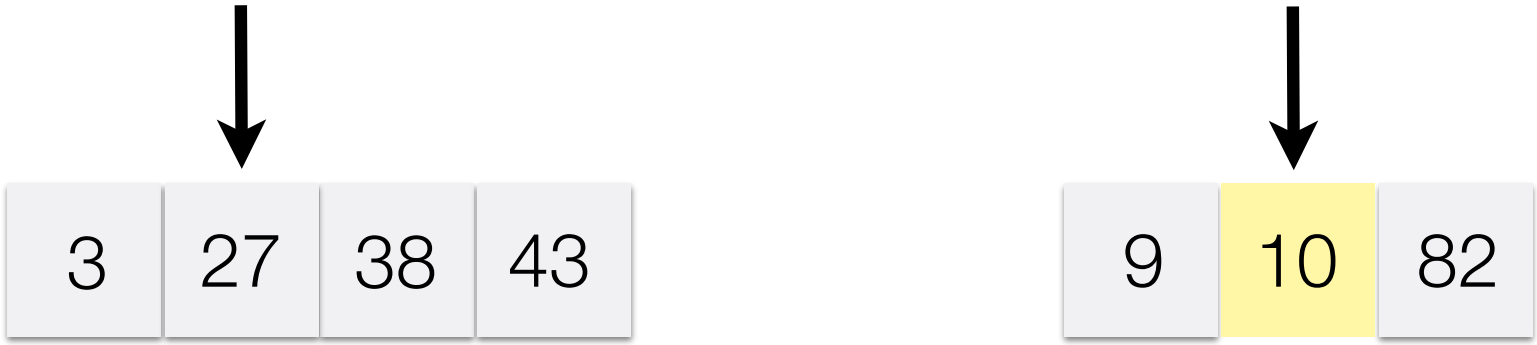
Merging Lists



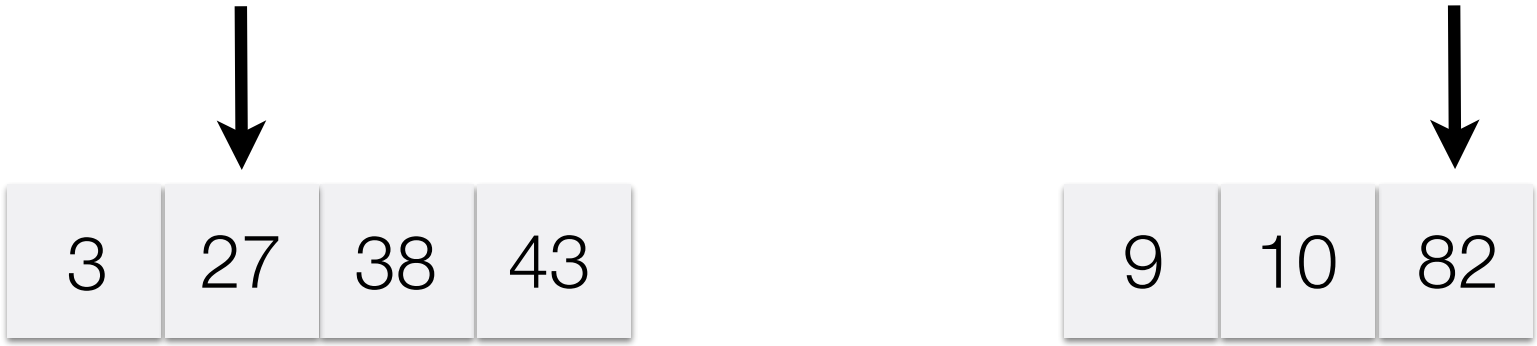
Merging Lists



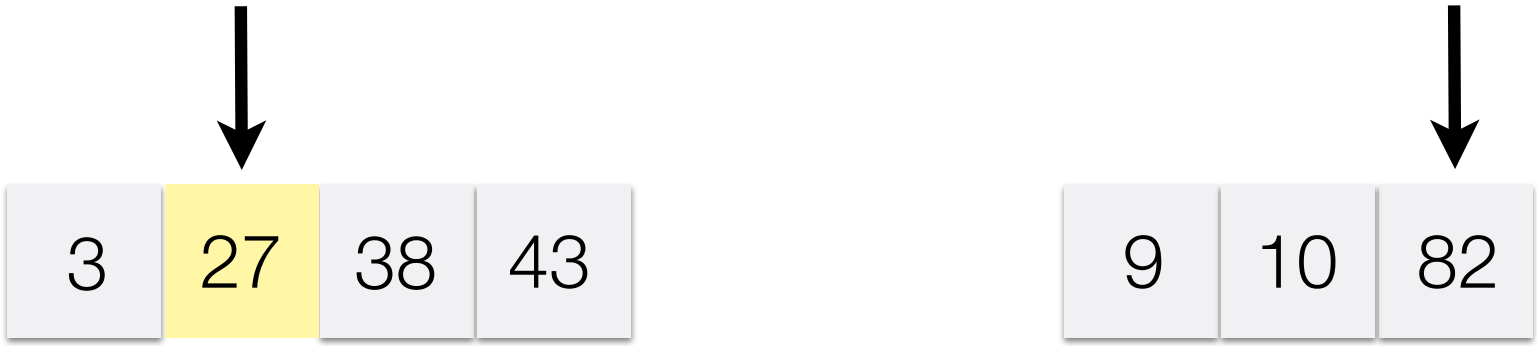
Merging Lists



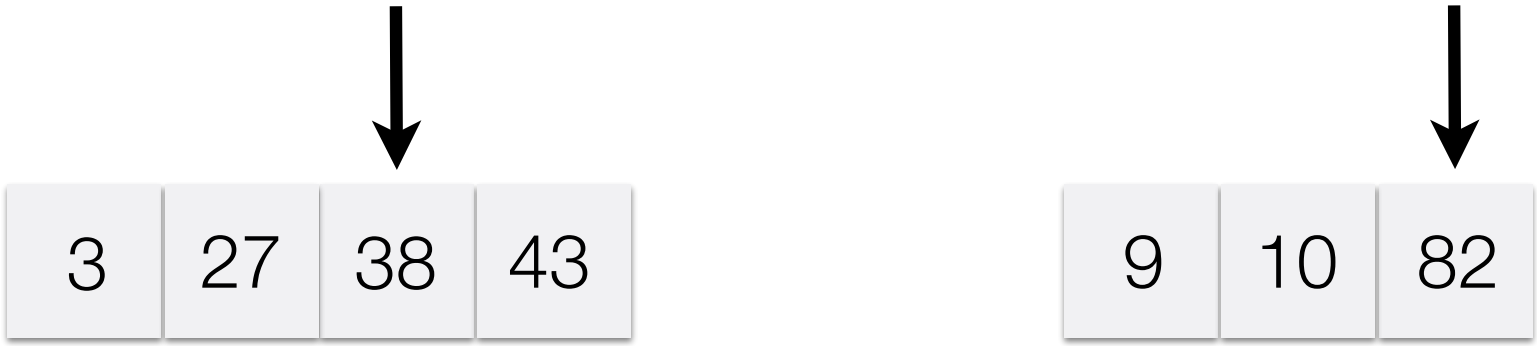
Merging Lists



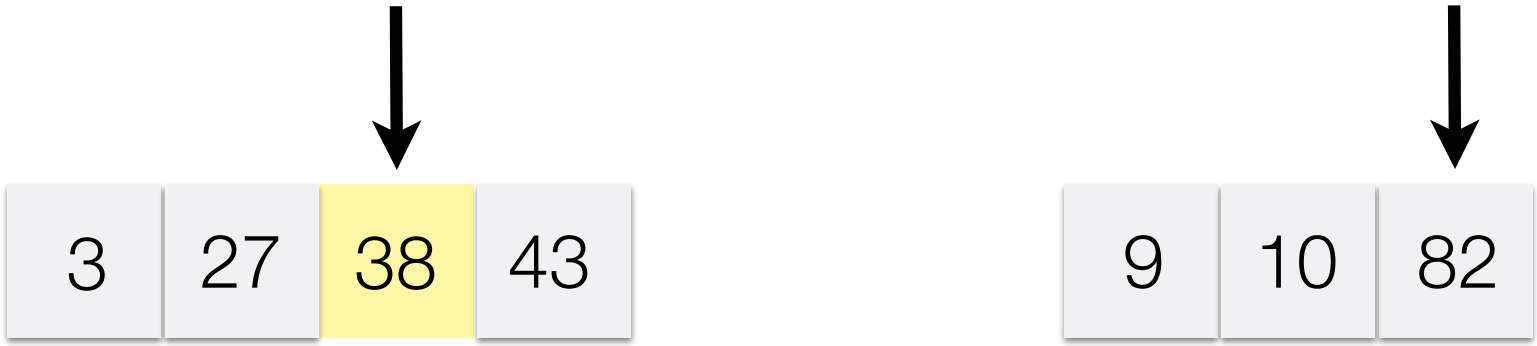
Merging Lists



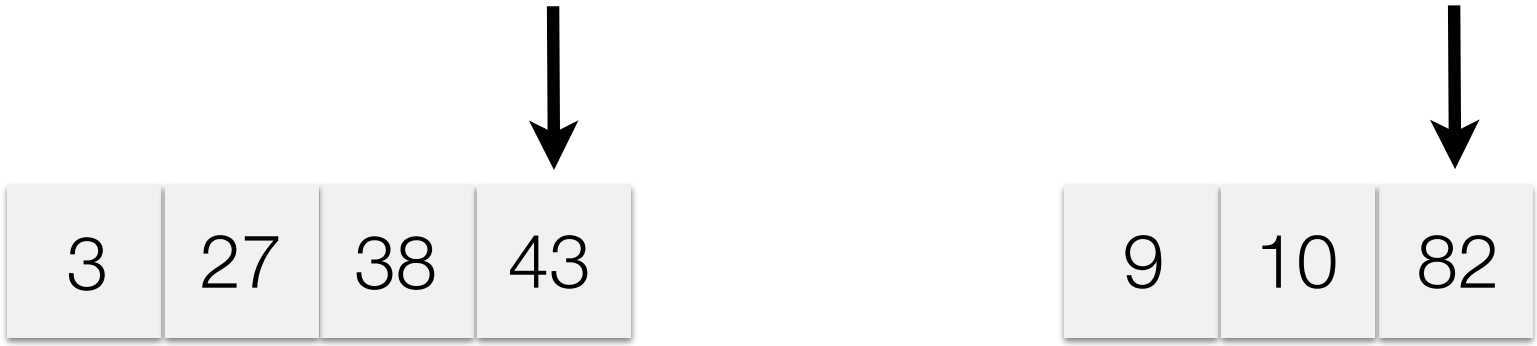
Merging Lists



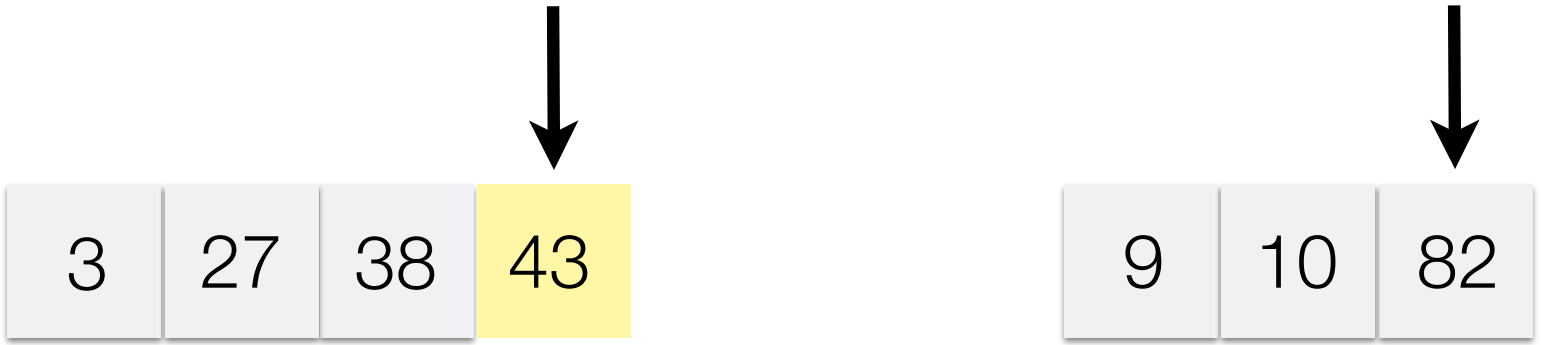
Merging Lists



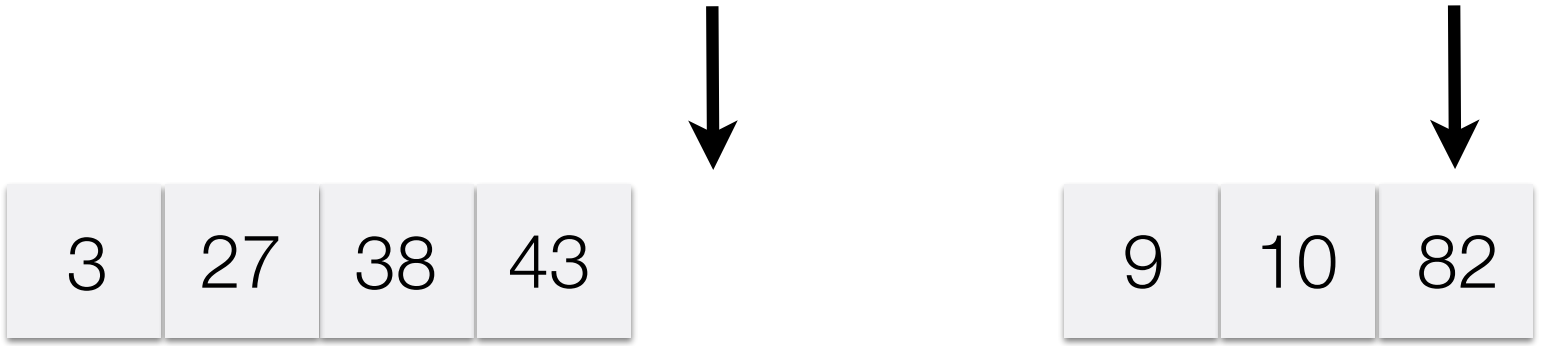
Merging Lists



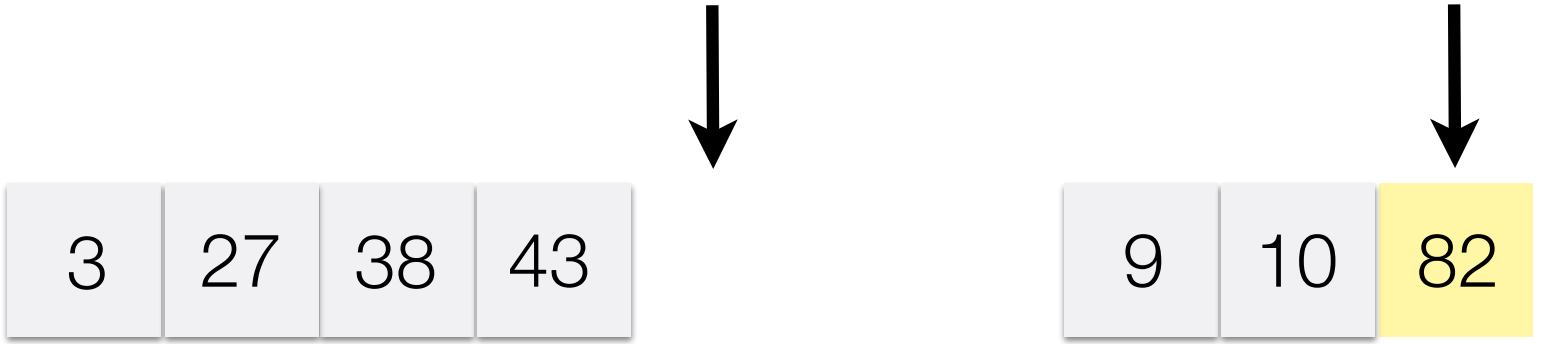
Merging Lists



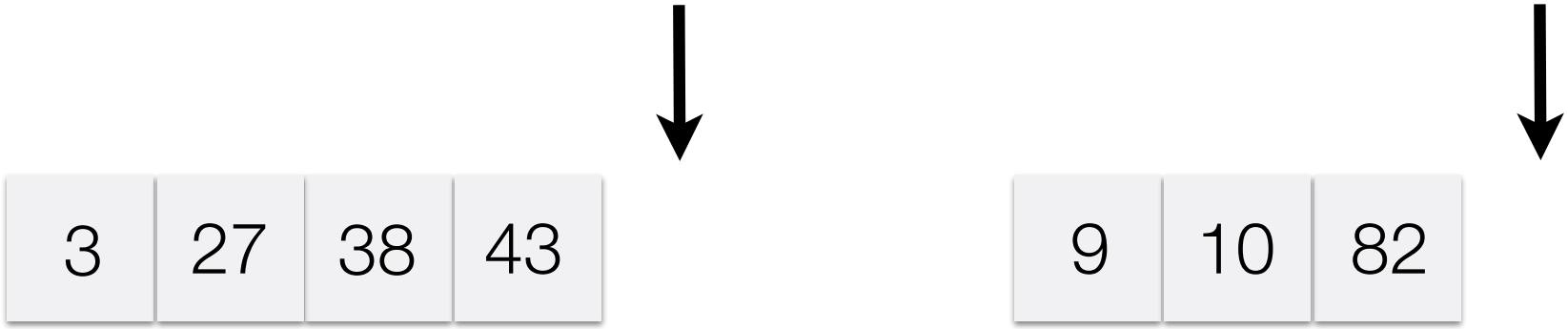
Merging Lists



Merging Lists



Merging Lists



Merging Lists

```
public static List<Integer> merge(List<Integer> a, List<Integer> b) {
  List<Integer> result = new ArrayList<Integer>();
  int i = 0, j = 0;
  while (i < a.size() && j < b.size()) {
    if (a.get(i) < b.get(j)) {
      result.add(a.get(i));
      i++;
    } else {
      result.add(b.get(j));
      j++;
    }
  }
  while (i < a.size()) {
    result.add(a.get(i));
    i++;
  }
  while (j < b.size()) {
    result.add(b.get(j));
    j++;
  }
  return result;
}
```

Merge Sort

- Base case
  - A single-element list is sorted
- Recursive call
  - Split the list into two halves, and merge-sort both independently
- Recombination
  - Combine the two sorted halves while maintaining (sorted) order

Merge Sort

```
public static List<Integer> mergeSort(List<Integer> list) {
  // Base case
  if (list.size() <= 1) {
    return list;
  }

  // Middle position in list
  int mid = list.size() / 2;

  // Recursive calls to sort left and right sublists
  // subList(s,t) is a sublist from s (inclusive) to t (exclusive)
  List<Integer> left = mergeSort(list.subList(0, mid));
  List<Integer> right = mergeSort(list.subList(mid, list.size()));

  // Merge the sorted sublists
  return merge(left, right);
}
```